Available online at www.sciencedirect.com
SciVerse ScienceDirect
Procedia Environmental Sciences 11 (2011) 193 – 199
Procedia Environmental Sciences

Resources Allocation in Virtualized Systems Based on Try-before-buy Approach

Ritai Yu,Congfeng Jiang,Xianghua Xu,Hui Cheng,Jian Wan

Grid and Service Computing Technology Lab Hangzhou Dianzi University Hangzhou 310037, China
jiangcongfeng@gmail.com

Abstract

In virtualization environments, resources are shared across multiple virtual machines(VMs), which results in contentions and even conflicting under heavily loaded or consolidated situations. In order to accommodate as many as service instances while still delivering performance guarantees, resource allocation should be optimized in a just adequate manner such that less resources will be utilized for a specific performance requirement. To achieve this goal, in this paper we propose a try-before-buy approach for allocating contending resources. This approach firstly does a try of minor resource allocation to find the performance feedback and search for the optimal amount of resource that should be allocated to the target virtual machine. This approach does not require a highly accurate performance model in a virtualized system where workloads usually change frequently with time in some intervals. Experiments on a Xen based virtualized environment are conducted and evaluated for its effectiveness. The results show that the proposed approach utilizes less CPU and memory resources to achieve the same performance goals compared to default Xen configuration with over-provisioning..

Keywords: Virtualization; Provisioning; Resource allocation; Performance feedback

1. Motivation

Virtualization has become a rapidly growing solution to large scale data centers and server systems for saving energy and operational costs. In virtualized environments, resources are shared across multiple virtual machines(VMs), which results in the contentions and conflicting under heavily loaded or consolidated situations [1]. In order to maximize the revenue and to accommodate as many as service instances while still delivering performance guarantees, resource utilization of a dedicated virtual machines should be minimized and the saved resources are restored in a resource pool for future use, either for new coming requests or new service instances, or just for energy saving purpose. Therefore, thin-provisioning and co-allocation of contending resources while providing performance assurance in a fair or prioritized manner to concurrent VMs is needed, since the resource capacity is finite for concurrent VMs.

doi:10.1016/j.proenv.2011.12.030

However, in order to allocate CPU, memory, and disk I/O resources properly and to provide adequate application performance guarantees efficiently, the following challenges must be addressed[2-4]:

(1) Automation. In virtualized autonomic systems, virtual machine monitor(VMM) is responsible for global resource allocation and the individual VM is responsible for its own application in a highly autonomic manner. A change in the resource demands of various VMs may require reconfiguring one or more VM allocations such that the system performance and Service Level Agreements(SLAs) based requirements can be guaranteed. It is also desirable to allocate resources among various VMs while still satisfying their SLAs requirements in response to the frequently changing data center conditions. Therefore, to provide real time performance assurance, all allocation decisions should be made automatically without human intervention, not manually.

(2) Adaptation. Traditional approaches to resource allocation are based on operating systems with full knowledge of and full control over the underlying hardware resources. However, the distributed nature of multi-layered virtualized environments makes such approaches insufficient. In this situation, it is difficult and sometimes impossible for workload characterization. Thus, the allocator should adapt to variations in workloads or system conditions and does not depends much on the workload characterization information.

(3) Scalability. In a highly consolidated environment, multiple VMs share multiple resources simultaneously, contentions and conflicting are unavoidable. For practical implementation, the resource allocation architecture should scale to a large scale virtualized environment with many applications and physical nodes by not requiring a single centralized controller or hardware capabilities.

All the above challenges make it impossible to directly apply traditional resource allocation techniques to the virtualization environment without modification. And the challenge is then to allocate appropriate resource to the upper level services and applications that have contending even conflicting resource demands.

In our approach, the VMM is responsible for allocating basic resources such as CPU slices, memory capacities, and disk I/O bandwidth. In runtime, resource allocation decisions are automated based on the *try-before-buy* methodology. In the *try-before-buy* approach, once the initial allocation is made, a *try*, i.e., a minor specific share of physical resource to a VM will be allocated automatically and the resulted specific performance feedback can be measured using application-specific performance metrics such as response time and/or throughput, to aid for the real *buy* decision, i.e., the next cycle of real allocation. Since the resource allocation decision is made only based on the estimation that the expected allocation can provide performance improvement when resource demand is increasing, it does not require much highly accurate knowledge about the relationship between application performance and resource allocation.

The remainder of this manuscript is organized as follows: In section 2, we propose the *try-before-buy* model and describe its mechanism for resource allocation. In section 3, we evaluate the algorithm in experiments, and we make extensive performance analysis. Finally we conclude our contribution in section 4.

2. Resources Allocation Based on Try-before-buy Approach

In this paper , we argue that the ultimate goal of resource allocation in virtualized environment is to provide predefined desirable performance guarantees with minimal resource utilization including CPU cycles, memory capacities, disk or network bandwidth, and energy. Or in other words, the ultimate goal of resource allocation in virtualized environments is to achieve maximal or higher performance under a predefined resource budget constraint.

Although there are many parameters and ways to represent the performance of a dedicated system or application, the response time is the most used and fundamental representative parameter. In real virtualized systems, the application performance, such as response time, is affected by various conditions including workload arrival rate and distribution, and resource allocated to the dedicated VMs. Significant

changes in the allocated resources can affect the application performance. In heavily multiplexed virtualized systems, the relationship between the resource allocation and application response time becomes more complicated. More importantly, this kind of relationship cannot be characterized by conventional mathematical models[5-9].

Therefore, in this paper we do not attempt to characterize the accurate real-time workload and performance modeling. In contrast, we use *try-before-buy* based resource allocation, which allocates resource step by step and does not need accurate performance model of the undertaken virtualized system. The *try-before-buy* approach can be summarized as follows:

Step1: The allocator uses the historical information of past performance and resource allocation for a given VM to approximate the resource demands in the future.

Step2: The allocator generates the amount of requested resources as well as performance requirements.

Step3: The allocator tries to regulate the resource allocation to the corresponding VM at a minor increase. If the performance gets better, the actual allocation will be initiated.

Step4: After the initiation stage, the allocator tries to keep allocating another amount of resource to the corresponding VM given the performance gets better.

Step5: The allocator stops to allocate resource to individual VM if the resulted performance keeps constant, or gets worse.

Since we do not need much data that far away from the current allocating period because the current and the future workload may not depend much on the previous workloads long time ago, we can use a relatively simple and highly efficient approach for performance modeling with light overheads. Linear regression is a simple regression method that attempts to model the relationship between two variables by fitting a linear equation to observed data. Here we use least squares regression to estimate the resource demand of individual VM. For every allocation interval, the allocator re-calculates a linear model that approximates the nonlinear and time-varying relationship between the resource allocation to VM and its normalized performance around the current operating point. The parameters can be re-estimated online using the recursive method. At the end of every allocation interval, the allocator collects the newly-measured performance value, normalizes it by the performance target, and uses it to update the values for the model parameters. Note that here we assume that variations in workloads that cause significant model parameter changes occur infrequently compared to the allocation interval, thus allowing the model to converge locally around an operating point and track changes in the operating point.

2.1 Try amount adaptation

In our *try-before-buy* approach, the resource demand only be satisfied when the allocation can make the system performance better, for example, to make the resource available earlier, or to make the workload finished earlier. In other words, more resources will be allocated only when better performance can be achieved, and less resource will be allocated only when there are less performance degradation after the allocation.

However, in heavily loaded cases where applications need large amount of resources urgently, the step-by-step *try-buy* approach cannot provide acceptable real-time performance due to the delay of performance response to newly resource allocations. To overcome this drawback , the try amount of the minor allocation should be adaptive to real time resource demand.

Here we use a normalized cost-performance ratio, i.e., *cpr*, to indicate the revenue of resource allocation to dedicated VMs. *cpr* is defined in the following:

$$cpr = \frac{performance\_increment}{resource\_increment} \quad (1)$$

In the above equation, resource increment and performance increment are percent values. When resource allocation is increasing, higher *cpr* represents better gained performance. When resource allocation is decreasing, higher *cpr* represents worse gained performance .

Here we use a simple performance feedback approach to regulate the try amount. For example, if the performance is improved after the first minor try of resource, the next try amount will be added with a *step-size amount*. It is same when we need to reduce the resource allocation to a specific VM and restore the resources to the resource pool. The real *step-size amount* can be decided in operation.

3. Experiments and Analysis

In order to evaluate our proposed approach under the impact of virtualized environments on different resource contention, we conducted experiments with some workloads sharing the same physical devices. To identify the beneficial of our approach, the investigated workloads include RUBiS[10], and a customized array manipulation program. These applications contains CPU, memory and disk IO requests which can be split into various components such as data access, index access, log writing, etc.

We developed a program to capture these parameters and to examine the behavior of the applications in a virtualized environment. All the experiments were conducted on a physical server, equipped with an Intel Dual-core i3 2.93GHz processors, 4GB memory, one Gigabit Ethernet cards and one 300GB 7200 RPM SATA hard disk. The machine is installed with CentOS 5.3 and the Linux kernel version is 2.6.18-xen-SMP. One VM images were built using the same distribution of the CentOS 5.3, and no changes were made to the kernel. Another VM is Windows XP SP2. In our implementation, the testing clients connected to the VMs using the network interface dedicated to the VMs. The allocator collected application performance statistics from the clients.

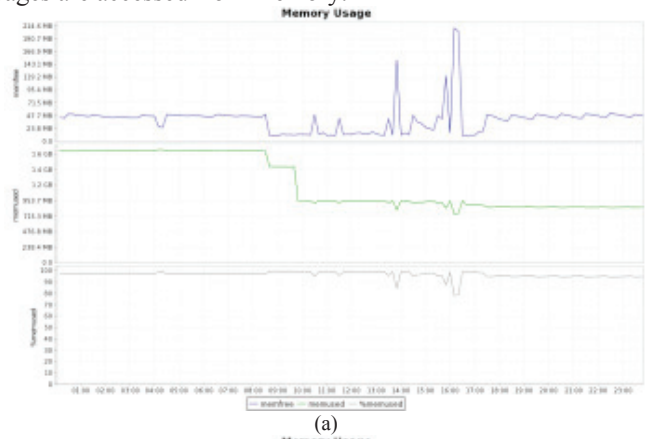
In the RUBiS benchmark, we used a dataset which includes 1 million customers, 20 million orders/month and 3 million products. We set the workload transactions at a very large scale and therefore the number of completed ones in our 15 minute test period is small. As time goes by, this results in higher throughput with higher access latencies. In case of interleaved random and sequential access, the latencies are improved and the improved latency can help handle bursts in workloads. Each user issues a transaction in a closed loop. The timeout for a transaction is set to 60 seconds. We noticed that 50 concurrent users were sufficient to keep the CPU fully utilized. With more users added, the overall latency increases, but the overall transactions per second did not increase much.

To implement the allocator's allocation decision, we use Xen's credit-based CPU scheduler which allows each domain to be assigned a cap. We used the cap to specify a CPU slice sharing for each VM to provide better performance isolation among applications running in different VMs. We evaluated our allocator in a number of experimental scenarios. In this section, we present the performance results from these experiments that demonstrate the effectiveness of the allocator. All the experiment results are listed in the Fig.1 and Fig.2. In the results, subgraph(a) stands for the results of default Xen[1], and subgraph(b) stands for our approach.

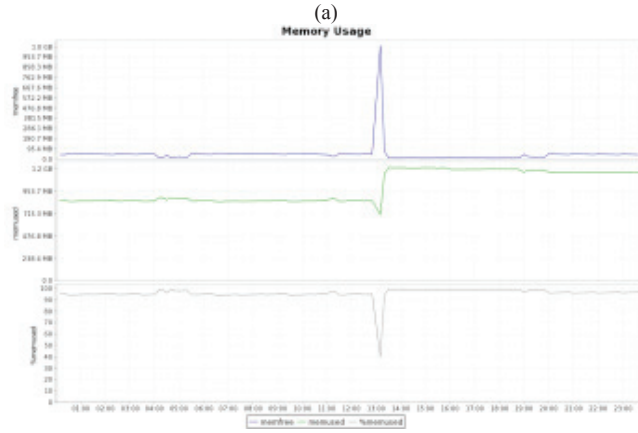
From the results we can see that our approach behaves well where there are two VMs residing in a physical machine, especially when the workload is relatively high. In some cases that all the VMs are extremely heavily loaded, the notification messages could not be sent to the allocator on time and thus resulted in higher latency. In our future work, we will add more additional functions to provide the allocator with higher reliability.

We found that as memory allocation reduces, memory-operations- per-second remains constant if the requested array hits in the VMs physical memory. However, due to the I/O latencies of paging, memory-operations-per-second reduces super-linearly as memory reduces further. In the implementation, the applications run in the default Xen settings, where a cap of zero is specified for the shared CPU on a node, indicating that the applications can use any amount of CPU resources. Note that the data of the first several

running is excluded to obtain steady state values. The application performs better with higher memory allocation since more pages are accessed from memory.



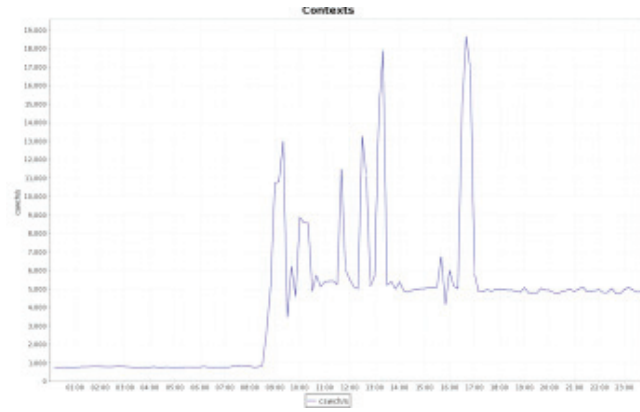
(a)



(b)

Fig.1 Memory usage

We also observed that contending disk I/O throughput has a strong relationship with the application's performance. For example, from low to medium levels of competing disk I/O, memory usage decreased substantially. Due to the page limitation, we only provide the memory usage and context switches statistics.



(a)

(b)

Fig.2 Context switches

4. Conclusions and Future Work

In data centers, virtualization provides the opportunity of carving individual physical servers into multiple virtual containers that can be run and managed separately. A key challenge is the simultaneous on-demand provisioning of shared resources to virtual containers and the management of their capacities to meet service quality targets at the least cost. We proposed a try-before-buy approach to solve this problem. Experimental results demonstrate that the proposed resource allocation approach can significantly reduce resource consumption while still achieving application performance targets.

In virtualized system, consolidation provides benefits of statistical multiplexing and helps in smoothing bursts in real workloads. However, simply putting random and sequential workloads together can also lead to performance degradation for the sequential workload without modifications. Our work is very helpful in enabling users to generate realistic impressions of real resource allocation approach for their research. And we believe that the proposed approach and techniques are useful for resource allocation among multiple VMs in virtualization environments. The simple solution can be applied to the current virtualization architectures and is easily usable by the system administrators of the data centers and cloud platforms.

In this paper, we do not consider the disturbance introduced into the system by the *try-before-buy* approach. The robustness will be extensively analyzed in our future work. We are also planning to propose a fine grained energy aware thin-provisioning approach for contending resource allocation with performance guarantees.

5. Acknowledgments

The funding supports of this work by Natural Science Fund of China (No. 61003077, 61003077,60873023 and 60973029), Technology Research and Development Program of Zhejiang Province, China (No. 2009C31033), State Key Development Program of Basic Research of China (Grant No. 2007CB310906), Natural Science Fund of Zhejiang Province (Y1101092), and Research Fund of Department of Education of Zhejiang Province (No. GK100800010) are greatly appreciated.

References

[1] Barham, P., Dragovic, B., Fraser, K., Hand,S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. Xen and the art of virtualization. In Proc. of the 19th ACM Symposium on Operating Systems Principles (SOSP'03), 2003, pp. 164–177.

[2] D. Carrera, M. Steinder, I. Whalley, J. Torres, and E. Ayguade. Utility based placement of dynamic web applications with fairness goals. In IEEE Network Operations and Management Symposium(NOMS'08), pp. 9-16, 2008.

[3] J. Xu, M. Zhao, J. A. B. Fortes, R. Carpenter, and M. S.Yousif. Autonomic resource management in virtualized data centers using fuzzy logic-based approaches. Cluster Computing, 11(3):213–227, 2008.

[4] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem. Adaptive control of virtualized resources in utility computing environments. In Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems(EuroSys 2007), pp.289–302, 2007.

[5] Jing Xu, Ming Zhao, Jose Fortes, Robert Carpenter, Mazin Yousif, "On the Use of Fuzzy Modeling in Virtualized Data Center Management," In proceedings of Fourth International Conference on Autonomic Computing(ICAC '07).2007. pp: 25 - 25 .

[6] Omesh Ticko，Ravi Iyer，Ramesh Illikkal，Don Newell .Modeling virtual machine performance: challenges and approache . ACM SIGMETRICS Performance Evaluation Review,Vol. 37(3), pp.55-60 ,2009.

[7] Sajib Kundu, Raju Rangaswami, Kaushik Dutta, Ming Zhao. Application Performance Modeling in a Virtualized Environment. HPCA2010 Proceedings of 2010 IEEE 16th International Symposium on High Performance Computer Architecture (HPCA2010), 2010 ,pp.1- 10.

[8] R. P. Doyle, J. S. Chase, O. M. Asad, W. Jin, and A. Vahdat. Model-based resource provisioning in a web service utility. In USENIX Symposium on Internet Technologies and Systems, 2003.

[9] M. N. Bennani and D. A. Menasc´e. Resource allocation for autonomic data centers using analytic performance models.In Proc. of ICAC2005, pp. 229–240. IEEE Computer Society, 2005.

[10] http://rubis.ow2.org/